

Rapport de stage

Stage - Analyse de logs applicatifs

INSA Lyon - Liris / EDF - R&D

Sébastien Gassmann

27 août 2008

Table des matières

1	Introduction	3
1.1	Etat de l'art	3
1.2	Présentation du stage	4
1.3	Contribution	4
2	Terminologie utilisée	4
3	DesEsper : Fonctionnalités voulues	5
3.1	Flux analysables	5
3.2	Détection de séquences	5
3.3	Reconnaissance des dates	5
3.4	Sorties	5
4	Benchmarks	5
4.1	Préambule	5
4.2	Réalisation des tests	6
4.3	Synthèse de ces expérimentations	10
5	Analyse et Conception de DesEsper	10
5.1	Analyse	10
5.1.1	Restriction sur les flux à analyser	10
5.1.2	Détection de la date	11
5.1.3	Permission sur les contraintes temporelles	12
5.1.4	Reconnaissance des motifs	12
5.1.5	Sortie des résultats	13
5.2	Conception	13
5.2.1	Fonctionnement de la partie analyse des évènements : Tâche "Analyse"	15
5.2.2	Fonctionnement de la partie sortie des résultats des sé- quences terminées : Tâche "Sortie"	16
5.2.3	L'interface graphique	16
6	Conclusion	16
7	Annexes	16

1 Introduction

Ce document finalise le stage qui s'est déroulé en été 2008 dans le laboratoire Liris en partenariat avec EDF R&D sur le sujet "Analyse de log applicatif d'une centrale hydraulique".

Partant au début sur une approche analyse de logs pour EDF, le stage s'est orienté au fur et à mesure vers une approche pré-analyse de flux. En effet, durant ce stage a été développé *DesEsper* un outil de pré-traitement d'analyse de flux d'événements.

1.1 Etat de l'art

Les outils qui permettent une analyse de flux ne sont pas nombreux et ont souvent des limites ne permettant pas d'effectuer les analyses souhaitées.

Grep

Grep [Fou05] est un outil ligne de commande unix permettant de rechercher les lignes contenant une expression régulière donnée. Il prend en entrée une liste de fichiers ou l'entrée standard et retourne les lignes respectant l'expression régulière sur la sortie standard.

SEC

SEC (Simple Event Correlator) [Rou04] est un outil à base de règles permettant d'étudier des flux d'événements. SEC prend en entrée plusieurs types de sources telles que l'entrée standard ou des fichiers texte. La lecture des événements se fait par l'intermédiaire d'adaptateurs (prédéfinis ou créés par l'utilisateur) à chaque mise à jour du flux. En sortie de SEC, des flux peuvent être également générés par les règles. Le chargement des règles se fait au démarrage de l'application à partir de fichiers texte. En interne, les événements issus des différentes sources sont réunies dans un même flux de données et estampillés par SEC. Ensuite, les règles sont appliquées séquentiellement sur ce flux d'événements.

Esper

Esper [Esp08] est une API Java open-source orientée ESP (Event Stream Processing) et CEP (Complex Event Processing). L'ESP est la problématique s'intéressant plus particulièrement à l'interrogation des flux de données à haut débit. Le CEP se focalise, quant à lui, sur l'extraction d'informations plus complexes dans des environnements générant des flux, tels que la détection de patrons d'événements complexes.

1.2 Présentation du stage

Dans le cadre de ma quatrième année à l'Institut National des Sciences Appliquées de Lyon dans le département Informatique, j'ai effectué un stage de 4 mois dans le laboratoire Liris sur le sujet "Analyse de log applicatif d'une centrale hydraulique".

Le stage se situe à la suite de l'étude effectuée par Frederic Flouvat [FPDR07] qui consistait à définir une architecture pour un logiciel qui permettrait d'analyser les logs. Cette étude fut menée en partenariat avec EDF R&D qui exprimait ses besoins et fournissait les logs.

Mon travail consistait à étudier les possibilités à utiliser un des systèmes étudiés par Frédéric Flouvat pour répondre aux besoins énoncés par EDF.

1.3 Contribution

En testant les systèmes étudiés par Frédéric Flouvat, j'ai rapidement été confronté à leurs limites en ce qui concerne la détection des temps. En effet, afin de détecter les séquences de démarrage et d'arrêt des groupes hydrauliques, il fallait que les dates soient extraites à chaque évènement de log reçu. Aucun système ne permettait de le faire.

J'ai donc eu l'idée de créer moi même un système d'analyse de logs qui répondrait parfaitement à mes attentes. C'est ainsi qu'est né le système *DesEsper* (le nom est dérivé du système d'analyse de flux *Esper*).

2 Terminologie utilisée

Un évènement : Une ligne dans un fichier ou un flux, cette ligne étant structurée.

Un fichier : Ensemble fini d'évènements.

Un flux : Ensemble infini d'évènements.

Une séquence : Suite chronologique d'évènements.

Une étape : Evènement d'une séquence correspondant à un ou plusieurs critères.

Un champ : Portion fixe d'un évènement.

Un critère : Expression régulière portant sur un champ d'un évènement.

Un groupe : Ensemble d'évènements qui ont un ou plusieurs critère en commun.

3 DesEsper : Fonctionnalités voulues

3.1 Flux analysables

Les flux analysables par le système DesEsper peuvent être finis (comme un fichier de logs) ou continus (comme des événements provenant en continu d'un autre logiciel).

3.2 Détection de séquences

La rôle principal de DesEsper est de pouvoir identifier une séquence définie par une suite d'évènement. Cette suite d'évènement est ordonnée chronologiquement.

La détection des séquences doit être basée sur la reconnaissance d'expression régulière dans un ou plusieurs champs de chaque événement de la séquence.

3.3 Reconnaissance des dates

Afin d'identifier une séquence chronologique il faut que le système puisse reconnaître et utiliser la date inscrit dans l'évènement. La définition d'une date dans un événement peut se faire de différentes manières, le système doit donc pouvoir s'adapter quelque soit les événements.

3.4 Sorties

Les sorties résultantes de la détection des séquences doivent être le plus configurables possible. En effet DesEsper est un outil de pré-traitement, la sortie doit être utilisable par un autre logiciel (logiciel d'analyse Data Minig, Tableurs, etc.).

4 Benchmarks

4.1 Préambule

Afin de comparer les 4 systèmes de l'état de l'art, ils ont été soumis à l'analyse de logs par de diverses séquences. Ce test est basé sur le benchmark effectué dans le document référence [FPDR07].

Le test a été effectué sur un ordinateur équipé d'un processeur Intel(R) Core(TM)2 Duo CPU T7250 @ 2.00GHz avec un total de mémoire de 2Go sur le système d'exploitation Ubuntu 8.04.

Les logs étudiés ont été générés par l'application SIMS et contiennent environ 1800000 événements, sauvegardés dans un fichier de 196Mo. Ces logs suivent le format défini dans l'API Log4C .

Exemple de logs :

```
3074214800 [2007-11-27 12:35:59,548] DEBUG sims - Run : channel manager : checking channel ages...
3074214800 [2007-11-27 12:35:59,548] DEBUG sims - Run : channel manager : channel ages were successfully checked
3051301776 [2007-11-27 12:35:59,622] INFO sims - PeriodicEvent : time reached
3051301776 [2007-11-27 12:35:59,622] INFO sims - Synchronisation : New synchronisation is running
```

Fichier de description de logs pour DesEsper :

```
<logs>
<description>
  <champ type="String" debut="0" taille="10">Numero</champ>
  <champ type="Date" debut="11" taille="25" format=" YYYY mm dd hh ii ss xcl ">Date</champ>
  <champ type="String" debut="37" taille="10">TypeEvt</champ>
  <champ type="String" debut="50" taille="0">description</champ>
</description>
<groupes>
  <groupe nom="tous">
  </groupe>
</groupes>
</logs>
```

Afin d'écrire cette description des champs de logs, il a fallut compter les positions des caractères pour chaque champs. Cette étape n'est pas très intuitive et peut facilement amener à des erreurs.

4.2 Réalisation des tests

R1 : Sélectionner tous les événements d'un log. [Cette requête donne 1 800 000 événements]

GREP : [1.09 sec]

```
grep -E '' ../logs/sims.log.2
```

Esper : [55.21 sec]

```
select * from BaseEvent
```

DesEsper : [42.61 sec]

```
<sequences>
<sequence nom="AfficherTous" unique="faux">
  <etape source="tous" timeout="0">
    <critere champ="description"></critere>
  </etape>
</sequence>
</sequences>
```

SEC : [2 min 51 sec]

```
type=Single
ptype=RegExp
pattern=.*
desc=$0
action=write ../sortie.txt $0
```

R2 : Sélectionner les événements contenant "err" (indépendamment de la casse) dans leur description. [Cette requête donne 144 événements] *Remarque* : Grep et Sec ne permettent pas de rechercher dans un champ particulier de l'évènement, ils recherchent dans l'évènement entier

GREP : [23.46 sec]

```
grep -E -i ' err' ../logs/sims.log
```

Esper : [17.38 sec]

```
Select * from BaseEvent where description regexp '.*(err|ERR|Err).*
```

DesEsper : [18.84 sec]

```
<sequences>
<sequence nom="AfficherErr" unique="vrai">
  <etape source="tous" timeout="0">
    <critere champ="description">(?)err</critere>
  </etape>
</sequence>
</sequences>
```

SEC : [71.93 sec]

```
type=Single
ptype=RegExp
pattern=err.*| ERR.*
desc=$0
action=write ../sortie.txt $0
```

R3 : Sélectionner les événements ayant eu lieu le 2007-11-26 entre 07 :12 :48 et 07 :25 :01. [Cette requête donne 1908 événements]

GREP : [34.41 sec]

```
grep -E -i '2007-11-26 07:((12:4[89])|(12:5[0-9])|(1[3-9]:[0-5][0-9])|(2[0-4]:[0-5][0-9])|
```

Esper : [9 sec]

```
Select * from BaseEvent where timestamp regexp '.*(2007-11-26 07:((12:4[89])|(12:5[0-9])|
```

DesEsper : [14 sec]

```
<sequence nom="AfficherIntervalTemp" unique="vrai">
  <etape source="tous" timeout="0">
    <critere champ="Date" formatDate="YYYY-mm-dd hh:ii:ss" comparateur='sup'>2007-11-26 07:1
    <critere champ="Date" formatDate="YYYY-mm-dd hh:ii:ss" comparateur='inf'>2007-11-26 07:2
  </etape>
</sequence>
```

R4 : Sélectionner les 10 derniers événements. [Cette requête donne 10 événements]

Grep : Impossible en utilisant grep. Par contre, la commande tail peut être utilisée.

```
tail -n 10 ../logs/sims.log.2
```

Esper : Impossible à partir du langage de requête mais l'API permet d'avoir accès aux derniers événements passés.

DesEsper : Impossible : pas de notion de début et de fin dans un flux. Pour un fichier : impossible , mais facilement implémentable

SEC : Impossible car SEC ne permet pas de détecter le dernier événement (mais il permet à partir d'un événement déclenchant une règle de retrouver les événements qui le précédent).

R5 : Compter le nombre d'événements contenant le patron de chaîne de caractères "Failed to call * service" (où * peut être n'importe quel nom). [1 valeur enregistrée : '47']

Grep : [21 sec]

```
grep -E -i -c 'Failed to call .* service' ../logs/sims.log.2
```

Esper [12.22 sec]

```
select count(*) from BaseEvent where description regexp '.*Failed to call .* service.*'
```

DesEsper [15 sec]

```
<sequences>
<sequence nom="AfficherIntervalTemp" unique="vrai">
  <etape source="tous" timeout="0">
    <critere champ="description">Failed to call.*service</critere>
  </etape>
  <sortie>NumeroRequete</sortie>
</sequence>
</sequences>
```

SEC [49,32sec]

```
type=Single
continue=TakeNext
ptype=RegExp
pattern=Failed to call .* service
desc= $0
action= eval %a (++$b); write ../sortie.txt %a
```

R6 : Déclencher une alarme si le nombre d'échecs de connexion (les événements contenant "ERROR" et "Failed to call * service") pour un même service est supérieur (ou égal) à 4.

Grep Impossible car grep ne permet pas de regrouper des événements en fonction d'une partie de leur description.

Esper Impossible car Esper ne permet pas de regrouper des événements en fonction d'une partie de leur description.

DesEsper Impossible car DesEsper ne permet pas de regrouper des événements en fonction d'une partie de leur description.

SEC Impossible de gérer un nombre indéterminé de compteur en parallèle.

R7 : Sélectionner les événements "error" (ou "err") de deux sources différentes associés à la même adresse IP.

Grep Impossible car grep ne permet pas d'exprimer des conditions entre plusieurs sources.

Esper Impossible car Esper ne permet pas d'appliquer une condition d'égalité entre deux parties de champs texte. En effet, il est seulement possible de rechercher un motif prédéfini.

DesEsper Impossible car DesEsper ne permet pas d'appliquer une condition d'égalité entre deux parties de champs texte. En effet, il est seulement possible de rechercher un motif prédéfini.

SEC Impossible de dissocier les événements des différentes sources si cette information n'est pas présente dans la description des événements.

R8 : Calculer le temps moyen d'une tentative de connexion ayant conduit à un échec, i.e. calculer le temps moyen de la séquence <Attempt 1, Attempt 2, Attempt 3, Error> avec un intervalle maximum de 2 sec entre chaque événement.

Grep Impossible car grep ne permet pas de détecter des séquences.

Esper Impossible simplement car Esper ne permet pas d'extraire les estampilles temporelles. Il faut pour cela développer un programme Java qui sera ensuite appelé dans une requête.

DesEsper [10,23sec]

```
<sequences>
<sequence nom="Afficher" unique="vrai">
  <etape source="tous" timeout="2000">
    <critere champ="description">attempt 1/3</critere>
  </etape>
  <etape source="tous" timeout="2000">
    <critere champ="description">attempt 2/3</critere>
  </etape>
  <etape source="tous" timeout="2000">
    <critere champ="description">attempt 3/3</critere>
  </etape>
  <sortie>DureeTotale</sortie>
</sequence>
</sequences>
```

SEC Impossible simplement. Il faut pour cela définir 4 règles pour détecter la séquence et développer un programme Perl pour extraire les estampilles de début et de fin, puis calculer la moyenne.

4.3 Synthèse de ces expérimentations

Le système DesEsper n'apporte pas la solution à tout les problèmes soulevés par les autres systèmes. Orienté gestion des dates, il permet d'exprimer plus facilement certaines requêtes (requête R3) lorsqu'il s'agit de manipuler les dates et de résoudre les problèmes de gestion des durées entre différents événements (Requête R8).

A requêtes et sorties égales, DesEsper est légèrement moins rapide qu'Esper, mais le système de reconnaissance d'expression régulièrement est encore en développement (Voir sections ci-dessous).

Le problème soulevé par les requêtes R6 et R7 n'est abordé dans aucune des solutions logiciels testés. Ces requêtes font appel à une détection d'éléments inconnus mais communs à 2 événements. Ces éléments correspondent à une expression régulière le plus souvent mais ne sont pas accessibles directement une fois l'expression régulière satisfaite. Ceci pourra être une étude à part afin de créer un algorithme de détection de ce genre de séquence.

5 Analyse et Conception de DesEsper

5.1 Analyse

5.1.1 Restriction sur les flux à analyser

Afin de pouvoir détecter les dates dans les événements, l'évènement doit pouvoir être découpé en deux champs minimum : un champ « Date » et un autre. Les champs sont définis par leur position dans l'évènement et leur taille. Afin de prévoir les champs à tailles variables, la taille du champ peut être définie

comme étant jusqu'à la fin de l'évènement (fin de la ligne). Ce découpage en plusieurs champs fixe est utilisée dans plusieurs systèmes d'analyse de flux tel que Cayuga (Référence [BDG⁺07]).

Le nombre de champ est illimité, mais un champ doit toujours se situer à la même position d'un évènement à l'autre sur une ligne.

Avant de détecter des séquences dans le flux, il est possible de grouper les évènements dans différents groupes selon des critères sur les champs définis. Ainsi il sera possible de détecter les séquences sur un type d'évènement précis, d'améliorer la rapidité de détection et la lisibilité des séquences.

Exemple d'évènement analysable (log EDF)

	APP.G1	03/06	07H00'06,9"	MARCHE	POMPE	REGUL.
	APP.G1	03/06	07H00'06,9"	FREINAGE EN		COURS
*ALARME	APP.G1	04/06	07H00'21,7"	DEFAUT	VANNE	de TETE
	Dis.G1	03/06	07H01'05,0"	MARCHE	POMPE	VANNE de TETE

Dans cette exemple d'évènements, le découpage peut être fait suivant quatre champs : un premier champ "Alarme", un deuxième champ qui correspond à un type d'évènement, un troisième champ "Date" contenant et un quatrième champ contenant la description de l'évènement.

Description XML des champs pour DesEsper

```
<logs>
<description>
  <champ type="String" debut="0" taille="10">Numero</champ>
  <champ type="Date" debut="11" taille="25" format=" YYYY mm dd hh ii ss xcl ">Date</champ>
  <champ type="String" debut="37" taille="10">TypeEvt</champ>
  <champ type="String" debut="50" taille="0">description</champ>
</description>
</logs>
```

5.1.2 Détection de la date

Les informations de la date doivent contenir au minimum le jour, le mois, l'heure, la minute et la seconde. Si l'année n'est pas renseignée, l'année en cours est assignée à l'évènement. La précision de la date peut aller jusqu'à la milliseconde. Le format de la date peut être très varié : l'année peut être définie par 2 ou 4 chiffres, le jour par 2 chiffres (01 à 31), le mois par 2 chiffres (01 à 12), l'heure peut être de précision variable (seconde, dixième, centième ou millièmme).

Exemple de date

APP.G1	03/06	07H00'06,9"	MARCHE	POMPE	REGUL.
--------	-------	-------------	--------	-------	--------

Description du format pour DesEsper

YYYY mm dd hh ii ss xcl

La date extraire de l'évènement ci-dessus : 03/06 07H00'06,9" peut être extraite facilement par le système DesEsper, en lui spécifiant les informations disponibles. Cette date contient les informations nécessaires : jour, mois, heure, minutes, secondes et facultatives : dixième de secondes.

5.1.3 Permission sur les contraintes temporelles

L'ordre des événements dans le flux peut être différent de celui défini par l'estampille temporelle de l'évènement. Pour cela, un temps de retard accepté peut être défini ainsi qu'une limite d'évènements autorisés à être en retard. L'évènement arrivant en retard (selon le retard accepté et la limite d'évènements permise) sera automatiquement replacé dans son contexte temporelle. Les événements, une fois arrivé dans le système, sont placés dans un buffer trié chronologiquement. Plus le temps de retard et la limite d'évènement définies seront grands, moins la vitesse d'analyse des séquences sera importante.

5.1.4 Reconnaissance des motifs

Les motifs reconnus par DesEsper sont définis sous forme de séquence d'évènements qui doivent chacun correspondre à des critères (correspondance de champs à des expressions régulières, restriction sur les dates, etc.)

Les champs sur lesquels sont appliqués les critères sont ceux définis dans la description des événements. Un champ « Log » est automatiquement défini, contenant l'ensemble de l'évènement.

Une contrainte temporelle maximale peut être définie entre deux événements consécutifs, ainsi un motif ne sera reconnu que si les événements arrivent dans un laps de temps imparti. La définition d'une contrainte temporelle minimale est en cours de développement.

A une séquence d'évènements définie est associé un nom et un état unique ou non, le nom sert à identifier la séquence détectée et l'attribut unique à permettre ou non la séquence d'être reconnue plusieurs fois en même temps (si un événement peut être commun à deux détections d'une même séquence)

Exemple d'expression d'une séquence

Exemple de séquence à détecter Une séquence de 3 événements : une erreur de connexion entre chaque événement un temps de 2 secondes maximum est autorisé, le premier événement doit contenir "attempt 1/3", le deuxième événement doit contenir "attempt 2/3" et le troisième "attempt 3/3".

Traduction de la séquence en XML

```
<sequence nom="Erreur connexion" unique="vrai">
  <etape source="tous" timeout="2000">
    <critere champ="description">attempt 1/3</critere>
  </etape>
  <etape source="tous" timeout="2000">
```

```

    <critere champ="description">attempt 2/3</critere>
  </etape>
  <etape source="tous" timeout="2000">
    <critere champ="description">attempt 3/3</critere>
  </etape>
</sequence>

```

5.1.5 Sortie des résultats

Les résultats d'analyse sont dirigés par défaut sur la sortie standard, mais peuvent être redirigés vers un fichier ou un flux. Le format de sortie est complètement configurable, un format de sortie est défini pour chaque séquence définie. Les informations de sortie suivantes sont disponibles pour le moment : premier événement détecté, dernier événement détecté (de la séquence), nom de la séquence, durée totale de la séquence, numéro de la séquence (Nombre de fois que la séquence a été trouvée depuis le début du flux), et pour chaque événement : nom des différents champs définis plus « Date » et « Log », durée entre le premier événement et l'évènement en cours, durée entre l'évènement précédent et l'évènement en cours, numéro de l'étape dans la séquence.

Exemple de sortie désirées Afficher pour chaque séquence détectée, le nom de la séquence, suivi de la durée totale et de la date de chaque évènement.

Traduction de la sortie en XML

```

<sortie>
NomSequence a durée DureeTotale
Voici les dates des évènements qui sont apparus :
:E::C:Date:C::E:
</sortie>

```

5.2 Conception

Fonctionnement général DesEsper analyse un flux d'évènements suivant des requêtes continues (séquences) définies par l'utilisateur. Ces requêtes s'appuient sur le découpage en champs de l'évènement et l'identification de la date au sein de l'évènement. Une séquence à détecter est constituée d'une ou plusieurs étapes (une étape correspondant à l'arrivée d'un évènement précis). Le résultat des séquences détectées est fourni en continue sur la sortie standard ou redirigé vers un flux. [Voir Figure 1]

Organisation en classes

Organisation des séquences en classes

- Une séquence = une instance de la classe Sequence
- Chaque Sequence contient une liste d'Etape
- Chaque Etape contient une liste de Critere

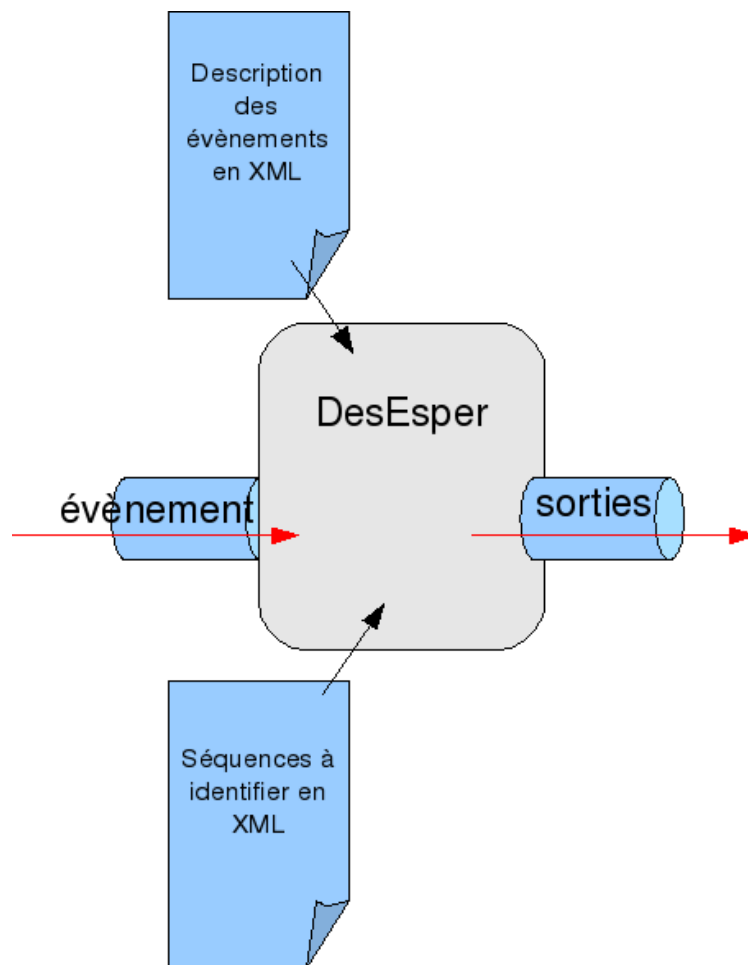


FIG. 1 – Schéma général du fonctionnement de DesEsper

Organisation de la description d'évènement

- Une description d'évènement = une instance de la classe Evts
- Chaque Evts contient une liste de Champ et une liste de Groupe
- Chaque Groupe contient une liste de Critere

Organisation des séquences en cours

- Une séquence en cours = une instance de la classe SequenceEnCours
- Les évènements sont manipulés sous forme d'Object Java Map

Un système multitâche DesEsper est un système multitâche, découpé en deux tâches (une tâche ayant pour rôle d'analyser les évènements au fur et à mesure qu'ils arrivent et une autre ayant pour rôle de fournir le résultat de sortie pour chaque séquence terminée). Une priorité plus importante est donnée à la première tâche afin de permettre une analyse plus rapide des évènements dans le cas où ils arriveraient en continue.[*Voir figure 2*]

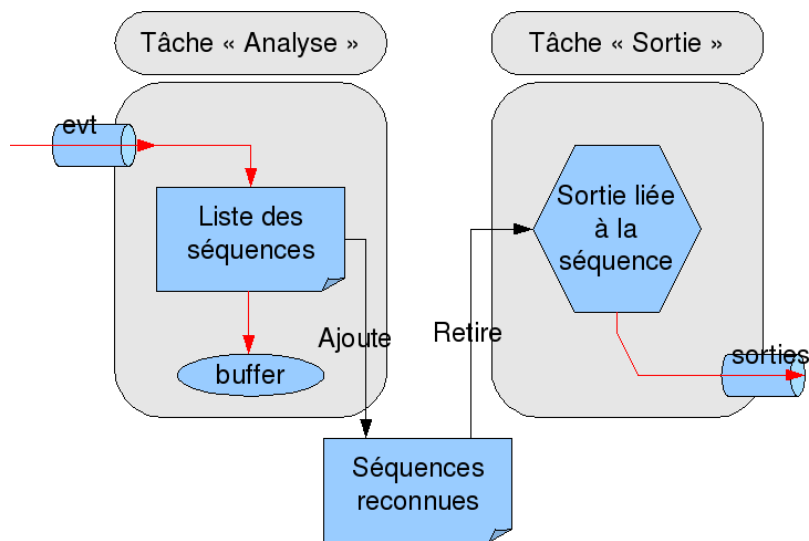


FIG. 2 – Schéma du fonctionnement multitâche de DesEsper

5.2.1 Fonctionnement de la partie analyse des événements : Tâche "Analyse"

Soit :

- une liste de séquences "en cours" (au début vide)
- une liste de séquences "de base" (définies par l'utilisateur)
- une liste de séquences "terminées" (résultant des séquences en cours, objet partagé avec la tâche "sortie")
- un buffer d'évènements dont la taille est paramétrable : deux paramètres : nombre d'évènement maximum et durée maximum entre le premier et le dernier évènement.

A chaque réception d'un évènement, la liste des séquences en cours est parcourue pour chercher celles que l'on peut avancer d'une étape grâce à cet évènement.

Ensuite, avec ce même évènement, la liste des séquences de base est parcourue pour chercher celles que l'on peut ajouter dans la liste des séquences en cours grâce à cet évènement.

Enfin, l'évènement est ajouté dans le buffer d'évènement, si cet évènement est en retard (c'est-à-dire si la date inscrite dans l'évènement est antérieure à la date de l'évènement le plus récent), les évènements arrivés après (au point de vue de la date des évènements) sont réinjectés dans le cycle (parcours des séquences en cours et de base).

Les séquences terminées (si toutes les étapes ont été franchies) et des requêtes expirées (si une étape a atteint son "timeout" sans recevoir l'évènement qui la ferait passer à l'étape suivante) sont enlevées de la liste des séquences en cours.

Tout le système de durée, moment, timeout est géré à partir des dates des événements, à aucun moment, le système ne se base sur le moment d'arrivée de l'évènement ou le moment actuel.

5.2.2 Fonctionnement de la partie sortie des résultats des séquences terminées : Tâche "Sortie"

La tâche "sortie" permet de séparer les créations des résultats des séquences détectées lors de l'analyse. Lorsqu'elle est active (du temps processus est libre), la liste des séquences terminées est parcourue et leurs actions sont effectuées pour chacune (Calcul des durées, affichage de champs, etc.).

5.2.3 L'interface graphique

L'interface graphique de DesEsper a été développé indépendamment. Son utilisation n'est pas absolument nécessaire. Une utilisation par la ligne de commande se fait tout aussi aisément.

L'interface graphique permet de visualiser directement à l'écran la sortie et de lancer et stopper l'analyse du flux simplement grâce à l'appuie de bouton.

6 Conclusion

Le stage est terminé mais le développement de DesEsper n'est pas terminé, de nombreuses améliorations peuvent y être apportées. L'ensemble des exemples, codes sources, documents sont disponibles à l'adresse <http://socq.liris.cnrs.fr/DesEsper/>.

7 Annexes

Manuel d'utilisation de DesEsper Fichier externe.

Rapport personnel Fichier externe.

Références

- [BDG⁺07] Lars Brenna, Alan Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald Mohit Thatte, and Walker White. Cayuga : A high-performance event processins engine. In *SIGMOD'07*, June 2007.
- [Esp08] EsperTech. Esper : Event stream intelligence, 2008. <http://esper.codehaus.org/>.
- [Fou05] Free Software Foundation. grep, print lines matching a pattern, 2005. <http://www.gnu.org/software/grep/>.
- [FPDR07] Frédéric Flouvat, Jean-Marc Petit, Pierre-Etienne Delon, and Alain Ribiere. Lot 2 : Proposition d'une architecture pour exploiter les traces générées par edf, March 2007.

- [Rou04] John P. Rouillard. "real-time logfile analysis using the simple event correlator (sec)". In *18th USENIX System Administration Conference (LISA '04) Proceedings*, November 2004. <http://www.cs.umb.edu/~rouilj/sec/>.